Assignment 4 – Calc Report Template

Arushi Tyagi

CSE 13S – Winter 24

Purpose

Audience for this section: Pretend that you are working in industry, and write this paragraph for your boss. You are answering the basic question, "What does this thing do?". This section can be short. A single paragraph is okay.

The program should mimic implementations of mathematical calculations such as the sine, cosine, tangent, and absolute value functions. It should also implement basic operations including addition, subtraction, multiplication, and division. With these functions, the program should be able to take a stack and perform the above operations with its values using Reverse Polish Notation (RPN). Essentially, the purpose of this assignment is to create a basic scientific calculator with all these features.

Questions

Please answer the following questions before you start coding. They will help guide you through the assignment. To make the grader's life easier, please do not remove the questions, and simply put your answers below the text of each question.

- Are there any cases where our sin or cosine formulae can't be used? How can we avoid this?
  If the value is too large or too small (not between 0 and 2 pi). Also, if the value is not an exact value like pi or pi/2, there may be a floating point error.

- What ways (other than changing epsilon) can we use to make our results more accurate? [1]
  We can use values that are multiples of pi, pi/2, 3pi/2 to make sure that we are getting the exact values. Also, if we make the numbers in a feasible range where they aren't too big or too small it would help as well.

- What does it mean to normalize input? What would happen if you didn't?
  By normalizing input, we map the angle to one between the range of zero and one. If we don't, our results may not be accurate or it may be difficult to make sure that they are accurate.

- How would you handle the expression 321+? What should the output be? How can we make this a more understandable RPN expression?
  The expression would be adding 3 and 2 first and then adding the sum of that with 1 which would result in 6. In RPN expression it would be 32+1.

- Does RPN need parenthesis? Why or why not?
  No because we are dealing with stacks where we can pop two numbers, do an operation and push the result back into the stack and continue.

[1] hint: Use trig identities

Testing

List what you will do to test your code. Make sure this is comprehensive. [2] Be sure to test inputs with delays and a wide range of files/characters.

- Text cases would be giving a basic expression as an input. For example doing 2 2 + would be a simple test case. I would create separate cases for addition, subtraction, multiplication, and division.
- A more advanced test case would be one with a more complex expression such as 22*3+5-1.
- We can also create a test case with negative numbers and a test case with the number 0.
- Another set of test cases would be some with division (both positive numbers, both negative numbers, one positive and one negative)
- We have to handle trigonometric functions (sine, cosine, tangent). I would also make one for absolute value.
- A test case with large numbers and one with very small negative numbers.
- Empty input.
- Invalid characters.
- Too big input (more tan the stack capacity which is 64)

How to Use the Program

Audience: Write this section for the user of your program. You are answering the basic question, "How do I use this thing?". Don't copy the assignment exactly; explain this in your own words. This section will be longer for a more complicated program and shorter for a less complicated program. You should show how to compile and run your program. You should also describe any optional flags or inputs that your program uses, and what they do.
The user would enter an expression in RPN format. They would enter a series of numbers and operations that the program will calculate. Then, the program will output the result and user can enter in another expression if they want to keep using the calculator.

Program Design

Audience: Write this section for someone who will maintain your program. In industry you maintain your own programs, and so your audience could be future you! List the main data structures and the main algorithms. You are answering the basic question, "How is this thing organized so that I can have a chance of fixing it?". This section will be longer for a more complicated program and shorter for a less complicated program.
There will be four main executables. The first one is operators.c which will have functions that perform basic arithmetic operations such as addition, subtraction, multiplication, and division. The second file is mathlib.c which will have functions implementing sine, cosine, tangent, and absolute value. Stack.c is the third file which will create the stack. The fourth file would be the calc.c file and would have the main function that would take the stack from stack.c and implement the operations and math functions from the first two files.

[2] This question is a whole lot more vague than it has been the last few assignments. Continue to answer it with the same level of detail and thought.

Pseudocode

Give the reader a top down description of your code! How will you break it down? What features will your code have? How will you implement each function.

- mathlib.c. In this file I will have five functions: Abs, Sqrt, Sin, Cos, and Tan. In the abs function, if the number is less than 0, indicating it's negative, I will multiply it by -1 to get it's absolute value. The sqrt function is already given. For the sine, cosine, and tangent functions, I will normalize the angle given so that it is between 0 and 2pi. Then, I will use the Mclaurin expansion and Taylor series formula to calculate the sine and cosine values. In the tangent function I will take the results from the sine and cosine functions and divide them.
- operators.c. The first function is apply_binary_operator which is responsible for taking two numbers from the stack and taking the operator from the input and calculating them. It will call both the stack function and also the operators function. Then, it'll return the result back to the stack function which will push it back into the stack. For checking if there are enough numbers in the stack, it will call the error checking function in calc.c. apply_unary_operator() is given already. Then, there is a series of operator functions that need to be written which will take in two numbers from the stack and perform addition, subtraction, multiplication, and division. The parse_double function is also already given.
- stack.c. This file is responsible for the stack ADT functions. In stack_push, An item will be pushed to the top of the stack by creating an array and making the top item of the array equal to the item. This will happen until the stack is full and as each item is adding, there will be a counter that will increment to indicate the size. For stack_pop, it will decrement a counter to keep track of the stack size. It will store the item that is being popped in a variable called item. In stack_peek(), this function takes into account two things. The first is whether or not the stack is empty or if the operation was successful. This returns a true or false. The second is the item pointer. It points to the top item's location in memory. This is where the variable is stored. stack_clear() just sets the counter for the stack size to 0. stack_print() prints the elements to stdout with spaces in between using printf and a for loop, looping through the size of the stack. (this is given)
- calc.c This file takes in the functions from the files mathlib.c operators.c and stack.c. This is how it works: There is a main while loop which loops until the end of the file (or expression in this case). We will go through the expression and separate them with their spaces. While there are still tokens to go thorugh, if the stack is not full, then it will check if the token is a number, binary operator, or unary operator. If it is a number then it will parse it and then push it into the stack. If it is a binary operator, it will pop two numbers from the stack and call the operator functions depending on if it is a + - * / or %. It will perform the corresponding calculation and push the result to the stack. If it is a unary operator, it will see if there is a number to pop form the stack and then perform the corresponding unary operator if it is a 's' 't' 'a' or 'r'. Then the result is pushed to the stack. If there is an error, it will print the corresponding error message. This could include if the stack is full, there aren't enough numbers for a unary or binary operator, or if there is an invalid character in the input. After calculating everything, it will call the stack_print() function to print the stack and the stack_clear() function to clear the stack and prompt the user for more input. If the user enters control d, the program will terminate.

Function Descriptions

For each function in your program, you will need to explain your thought process. This means doing the following

- The inputs of every function (even if it's not a parameter)
- The outputs of every function (even if it's not the return value)
- The purpose of each function, a brief description about a sentence long.

- For more complicated functions, include pseudocode that describes how the function works
- For more complicated functions, also include a description of your decision making process; why you chose to use any data structures or control flows that you did.

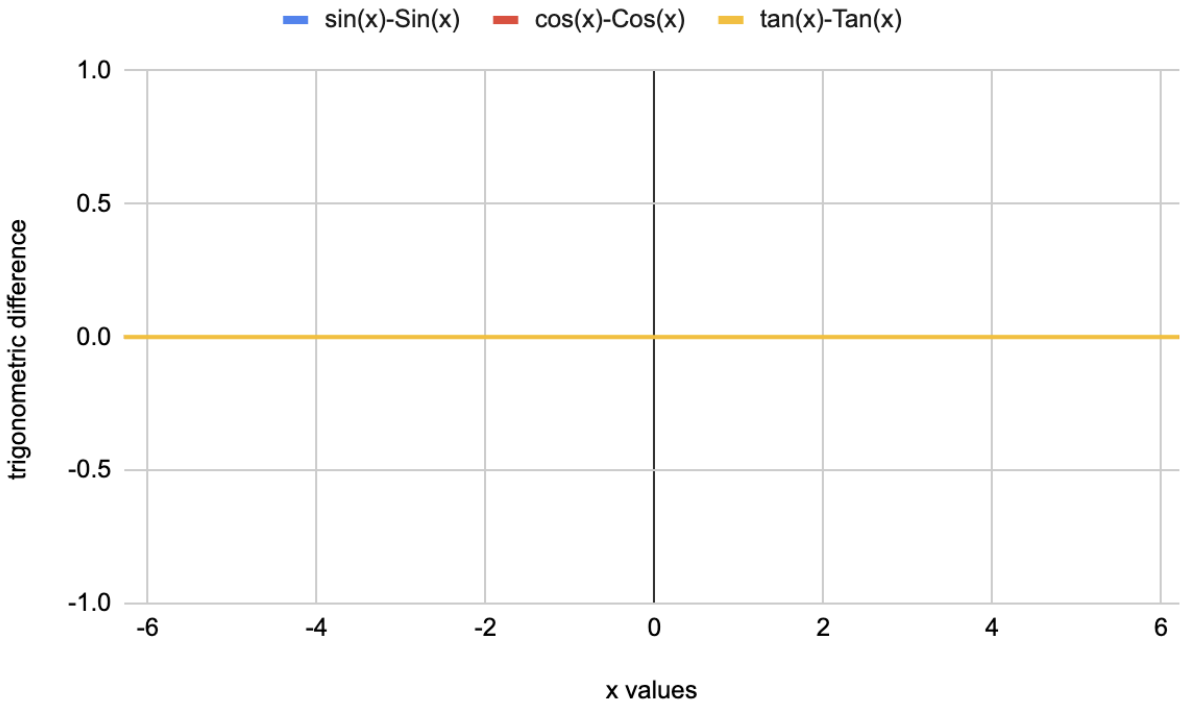
Do not simply use your code to describe this. This section should be readable to a person with little to no code knowledge. **DO NOT JUST PUT THE FUNCTION SIGNATURES HERE. MORE EXPLANATION IS REQUIRED.**

- Function: Abs. Inputs: x. Outputs: absolute value of x. Purpose: takes in a number and returns the absolute value of it.
- Function: Sqrt. Inputs: x. Outputs: square root of x. Purpose: take in a number and returns the square root of it.
- Function: Sin. Inputs: x. Outputs: sine value of x. Purpose: take in a number and returns the sine value of it.
- Function: Cos. Inputs: x. Outputs: cosine value of x. Purpose: take in a number and returns the cosine value of it.
- Function: Tan. Inputs: x. Outputs: tangent value of x. Purpose: take in a number and returns the tangent value of it.
- Function: apply_binary_operator. Inputs: binary operator function. Outputs: boolean (true or false) if operation is successful. Purpose: Takes the numbers from the stack and performs a mathematical operation.
- Function: apply_unary_operator. Inputs: operator. Outputs: boolean (true of false) if operation is successful. Purpose: Takes the numbers from the stack and performs a unary operation (sine, cosine, tangent, absolute value, square root)
- Function: operator_add. Inputs: left side doubles and right side doubles. Outputs: sum of both sided doubles. Purpose: Adds two doubles together.
- Function: operator_sub. Inputs: left side doubles and right side doubles. Outputs: difference of both sided doubles. Purpose: Subtracts two doubles together.
- Function: operator_mul. Inputs: left side doubles and right side doubles. Outputs: product of both sided doubles. Purpose: Multiplies two doubles together.
- Function: operator_div. Inputs: left side doubles and right side doubles. Outputs: quotient of both sided doubles. Purpose: Divides two doubles together.
- Function: parse_double. Inputs: string pointer location pointed by a pointer d. Outputs: boolean (true or false) if string is a valid number. Purpose: Takes a character and parses it into a precision floating double.
- Function: stack_push. Inputs: item to put into stack. Output: boolean (true or false) if stack is at capacity. Purpose: Pushes the item to the top of the stack.
- Function: stack_peek. Inputs: item pointer. Output: boolean (true or false) if stack is empty. Purpose: copies the first item in the stack to the item pointer.
- Function: stack_pop. Inputs: item pointer/address. Outputs: boolean (true or false) if stack is empty and does not need to be changed. Purpose: Takes an item out of the stack.
- Function: stack_clear. Inputs: none. Outputs: none. Purpose: clears the stack and sets the size to zero.
- Function: stack_print. Inputs: none. Outputs: none. Purpose: prints the stack.

- Function: main (in calc.c). Inputs: all the functions from the other 3 executables. Purpose: this function uses all the operators and math functions and stack functions to take an expression and calculate the result.

Results

Follow the instructions on the pdf to do this. In overleaf, you can drag an image straight into your source code to upload it. You can also look at `https://www.overleaf.com/learn/latex/Inserting_Images`



The code to create the graph data is given in graphing.c. There was no error that was shown, possibly because it was only rounded to 10 digits. Although only the tangent difference is visibly shown, the sine and cosine difference was also consistently at 0 for a range of x values form -6 to 6.

References